$1.75

July/August, 1982

DOSPLUS NEWS INFORMATION CENTER

M I C R O T E R M

More and more hardware and communications services are allowing speeds up to 1200 baud. Soon, some may be going faster than that. Today's terminal software simply can't keep up. But now there is an alternative. Micro-Systems Software introduces MicroTerm, the high speed terminal.

Model III MicroTerm will communicate, without insertion of null characters, at 4800 baud. Guaranteed. No cop-outs, no question. MicroTerm is so fast that you can exit from the terminal to the main menu, adjust video width, open the buffer, turn on the printer, or any one of dozens of other functions, and return to the terminal mode **without missing a thing!**

MicroTerm continues to input from the RS232, even while at the main menu. This is the only terminal capable of such an astounding feat. MicroTerm offers you most of the features that "Brand X" smart terminals have, plus it gives you: • Ultra high baud rate operation (up to 9600 in certain cases). • Input while at menu. • Easy to use translation tables. • Easy to use phone number listings. • Maximum auto dial support — most major brands. • Direct file transfer companion program included at no exta cost (compatible with DFT). • DOS commands from menu without exiting program. • Over 34K of capture buffer (in a 48K TRS-80). • Can be set to automatically dial telephone and transmit buffer at preset time without **any** operator intervention.

And many, many more great features, MicroTerm is so fast you must see it to believe it. The various menus are displayed so fast, they seem to jump out at you. Status of various functions can be displayed and altered in split seconds.

For the computerist who wants the ultimate, state-of-the-art terminal software, there is no other choice.

MicroTerm retails for $79.95, but registered DOSPLUS owners can purchase it for only $59.95. $20.00 off the retail price! MicroTerm comes complete with the terminal program, the direct file transfer program, some standard translation tables, and documentation.

Don't delay, order yours today! Specify when ordering: Model I or III and whether you want it on 40 or 80 track media. Requires a 16K TRS-80 with one disk drive. We recommend 48K for serious communications work. MicroTerm will be available beginning June 30, 1982.

MICRO-SYSTEMS SOFTWARE, INC.

4301-18 Oak Circle
Boca Raton, FL 33431
Telephone: (305) 983-3390
800-327-8724

Introduction

Welcome to the July/August issue of the DOSPLUS NEWS INFORMATION CENTER. This is the fourth issue of the newsletter and it is a good one. We've got lots of things lined up for you that I think you'll enjoy.

This issue's feature article is on Input@, the new BASIC input function added by DOSPLUS 3.4/4.0. Many of you have had questions on using it, I hope this will prove to be of aid.

Todd Tolhurst has returned with his popular column "Random Routines" (our machine language corner). This issue has a program called "Macro-Key" which does pretty much what the name says. It will allow you to enter many keystrokes at the touch of a single key. A very useful utility.

Our featured guest columnist this issue is Kyle Dunn of Dunn-Write Software, an outstanding local custom software house. Kyle writes about accuracy in BASIC programs and correcting rounding errors.

There is a patch for Scripsit 3.2 to enable saving with an exact end of file under DOSPLUS, a new product announcement about the Vehicle Management Program from Dunn-Write, and a BASIC program from Hile Troy (in Random Routines) that does a file to file compare (byte for byte).

This is unofficially a "BASIC programmer's issue". Between my article on Input@, Kyle's article on rounding, and Hile Troy's program, we are concentrating on the BASIC programmer this time. We hope that you will be able to profit from the information presented.

So you see, it IS a good issue. I know summer gets slow, but we get inspired! Don't miss a page.

Mark R. Lautenschlager
Editor

Input where?
or
How do I use this 'Input@' thing?

There has been a lot of controversy surrounding this command. This is one of the few commands in DOSPLUS that a user will write in and say "I sure think its neat, but I wish I knew what to do with it...". Well, every command in DOSPLUS has a purpose, or it never made it in. We have not yet implemented something in the system because it has no practical usage potential. Input@ is no different.

However, to properly explain Input@, I must first deal with two things :

(1) History of input routines.

(2) Programming techniques.

We will look at each of these in turn. But first, let me set the record straight on a particular subject. Yes, there IS a typo in all editions of the DOSPLUS 3.4/4.0 manual regarding Input@. In the manual it states the the prompt string may be a variable and that the type specifier string must be a literal. This is backwards. In fact, the prompt string MUST be a quoted literal while the type specifier string may be a string variable. You see, if a prompt string is not desired, that position may simply be omitted. Because of that, we would have a problem in detecting whether what you have entered is a prompt string variable or an invalid variable for the field length. So we restrict the prompt string to a literal.

History of input -

Way back in the dark ages (before Tandy said "Let there be expansion interfaces and disk drives"), we all used Level II BASIC. And there was only one way to have somebody input data into a variable, a little statement called "INPUT".

INPUT was neat because it meant that we could ask questions and expect reasonably sane answers from the operator. This was of course, adequate for the time. But shortly, problems began to appear. First, there was punctuation. You see, INPUT doesn't like any sort of punctuation that might be declared a delimiter to appear in the middle of your input. If one did, INPUT simply terminated it there.

The second problem was a question mark. INPUT always put one wherever it was asking for data. After a while, you began to wish that you could look at something else for a change. The third problem was also cosmetic in nature. I'm referring, of course, to INPUT's annoying habit of printing a carriage return at the end of the input field when you would press ENTER. This caused the line immediately underneath the line that you just pulled the input from to be erased. This REALLY punched a hole in any efforts to make truly controlled screen input.

The fourth problem with INPUT was that it was hard to visually show the user exactly how many characters they could input at a particular position and when they were running out of space. This was programmed around by the use of INKEY$, but we will get into that a little later. The final problem with input was that it did not automatically clear out the variable that you assigned it. For example, if A$ was equal to "TEST" and I used the statement :

```
INPUT A$
```

to which you replied by simply pressing ENTER, A$ would still be "TEST". You had no way of knowing whether they had typed "TEST" again or pressed ENTER. The only way around this was to zero out each variable before you input it, but that seemed like it should be done for you. But, INPUT was what we had, and to use it seemed wonderful.

Then came disk drives and DOS' and better yet, Disk BASIC! Now THIS was neat. WE had all kinds of new commands. And to top it off, we had a new input command called "LINE INPUT".

LINE INPUT was really neat in that it was free from a number of the aforementioned problems that INPUT had. First of all, it didn't care about punctuation. It would input a whole line of text from the keyboard just as it was typed in and let you decide what to do with it.

Second, the question mark was gone. LINE INPUT would allow you to print whatever character caught your fancy and then input after that point. It also automatically would zero the variable that was given to it to return the data in.

However, two of the more serious problems still remained. First, whenever you used LINE INPUT, when the input was complete and the operator pressed ENTER, the next line on the screen went away. Also, there was still the problem of visually showing the operator how much room he had for that particular entry and hand in hand with that came the problem of easily restricting the operator from over-typing a field to crash the program.

In light of all these, programmers turned to the INKEY$ routine. INKEY$ is a really neat command in BASIC that allows you to check the keyboard and return with the key being pressed (if any). It does all this without pausing the program and placing a prompt on the screen. But it only does it one character at a time. For example, if I used the statment :

```
A$=INKEY$
```

Whatever key was being pressed at the time that that statement was executed would be returned to me in the variable A$. This presented some possibilities to the programmer. Soon subroutines were cropping up all over to use the INKEY$ statement in a loop and pull in just the required amount of characters. Then they began to get fancy.

They would display a row of characters on the screen to show you the length of your field and then display each character as it was typed in. The advantages are plain. We no longer have the problem of the line beneath the input field being erased, because when your INKEY$ routine sends you a carriage return, you simply cease input and proceed. It also visually reinforced the field length and prevented the operator from crashing the program as easily.

But now we had one drawback that was more serious than all the others combined. Speed. These routines, especially the really smart (12 different control keys, etc.) and pretty ones, were getting very slow. Operators that weren't even particularly good typists were out-running the keyboard. Characters missed, tempers flared, and overall efficiency dropped.

Some programmers turned to machine language subroutines to duplicate what they had been doing in BASIC. This worked pretty well but then you had the problem of compatibility with all versions of BASIC. There were also many excellent BASIC programmers that didn't have the ability to write such subroutines.

The need became awesomely apparent to us here at Micro-Systems when we were designing DOSPLUS 3.4/4.0. We decided that a good alternative would be to give the BASIC programmer a new input keyword. In effect, incorporating these machine language subroutines into BASIC and doing the work for you. You call it directly from BASIC and never have to worry about interface problems. And so Input@ was born.

Input@ -

Input@ is the new input keyword that we provided for you in the DOSPLUS 3.4 and 4.0 series Extended Disk BASICs. It was designed primarily as a replacement for these INKEY$ subroutines and as an alternative to using LINE INPUT. Input@ allows you to utilize a highly controlled screen input routine with a minimum of effort.

An Input@ statment consists of potentially six items, with one being optional. Your general format is :

Input @ <pos>,"prompt",fl,"tf";rv$

Input @ is the DOSPLUS BASIC keyword that activates the Input@ function. The space between "Input" and "@" is optional and may be omitted.

<pos> is the screen postion that you wish to have the input area located. Be advised that if you are using a prompt string, this is the location that the prompt will be printed at and the input field will immediately follow. If no prompt string is specified, actual input will begin at this location. The space shown between "@" and "<pos>" is optional and may be omitted. The values for <pos> can run from 0 to 1023.

,"prompt" is the optional prompt string. If you are not using a prompt string, then you may completely omit this field (leading comma and all) and specify the length. If a prompt string IS specified, it will be printed at the location denoted by "<pos>" and the input field will follow immediately after the prompt string. If you desire a space between the prompt and the input field, be certain to insert one in the prompt string. This MUST be a quoted string literal.

,fl is the field length position. This may be either a constant or a variable. Whatever value you specify at this location will be used for the length of the field. Input@ will ingnore all fractional values and use the integer portion of any variable or constant used here.

,"tf" is the field type flag. This is used to indicate the types of input that will be allowed and to indicate whether or not you wish to automatically terminate input when the field is full. Please note that this can be expressed as either a string variable or a quoted literal. You may specify a "$" for alphanumeric input or a "#" for numeric input only. Including an "*" sets the return on full field option.

;rv$ is the return variable. This must be a string variable. This variable is that which Input@ will use to pass the keyboard input to your program. Please note that the return variable must be a string variable. In that respect, Input@ is the same as LINE INPUT (you may only input string data). Also note that this position is set off from the command line by a semi colon while the rest of them use commas. This IS required.

Input@ itself has its own special string space and you do not have to clear for it. However, you must have enough string space clear for Input@ to put your data into the return variable. For example, if you have the default amount of 50 bytes clear for string space and you use Input@ to input a 75 character long field, it will. Until you press ENTER. As soon as it takes those 75 characters from its field and attempts to put them into the return variable you have specifed, the program crashes with an "Out of string space" error.

Programming techniques -

The most important factor in program development is programming philosophy. The way you think a program should be written is the way that you will write it. No matter how advanced or wonderful a paricular feature is, unless you use it efficiently the value is lost.

We are talking about Input@, so I will confine my discussions to input routines. In any program, you have a certain set of variables that pertain to the data you are gathering. For example, if you were developing a mailing list, you might have "A$" equal to the name, "B$" equal to the address, etc. The way such programs have often been written in the past, these variables are actually used to capture the input from the screen. It is my contention that this is NOT such a good idea. For two reasons :

(1) Lack of error trapping.

(1) Lack of independant action without altering file variables.

There is a lack of error trapping because of the nature of the variables you will be dealing with. In most programs, there are string and numeric variables. If you input directly to your variables, then someone can crash your program by typing in a number too large for the variable type that you are using and achieve an overflow error.

By placing all input into a specially defined input string, you can safely interrogate it as to what sort of input you have and then set the numeric variable equal to the value of the string (using BASIC's VAL(string) function).

It is true that you can trap these overflow errors by using the ON ERROR GOTO facility, but that does not afford you the option of determining what sort of error was created. For example, did the operator enter a number that was too large or did they enter a valid control word such as "END" or "ABORT"? For this, and other reasons, I prefer inputting ALL data as strings first and then interrogating them for valididty before passing them to their actual permanent variables.

Also, when you input your data into an intermediate string first and have time to examine the input before deciding what to do with it, you may take effective independant action without changing the variables used for your files. For example, let's consider that mailing list again. If we are inputting the name from the screen directly into the variable used to store the name, and the operator only presses ENTER, that variable will be left set to "" (null string). That means if you are having them press ENTER to abort, you must do it at a point where there is no danger of wiping out currently allocated variables. This prevents "full screen" input and edit of data from being easily done.

That's were Input@ comes in. First of all, Input@ will never return with the null string. There are two ways to terminate input when using Input@. You may press either ENTER or CLEAR. Now, if ENTER or CLEAR is the only thing you press, Input@ will pack these into your return variable. This means that you have two potential control moves instead of just the one. This makes it easy to page up and down the screen.

In addition, this is one of the most common areas of misunderstanding with Input@. It was NOT designed to return you a null (empty) string. When we designed the command, we wanted to have two control keys available. Therefore, if you do not enter any characters, but simply press ENTER of CLEAR, Input@ will return a CHR$(13) for ENTER and a CHR$(31) for CLEAR. This lets you know two things: (1) no characters were entered and (2) what key was pressed.

If any characters at all are entered, the CHR$(13) and CHR$(31) will not be included in that string. So if you have valid data input to you, you need not fear that we have added any control codes to it.

To cover it easiest, I think a programming example is in order. The following listing of a program I call INPDEMO/BAS will illustrate one technique for using Input@.

INPDEMO/BAS

```
10 '    Sample program to demonstrate Input@
20 '    Programmer : Mark R. Lautenschlager
30 '    Created : 06/24/82
40 '    Updated :   /  /
50 '
60 CLEAR 1000 : DEFSTR S,V : DEFINT I-N
70 VL=CHR$(30) : VC=CHR$(31) : VR=CHR$(13)
80 '
90 CLS : PRINT "Demo program for Input@ questions"
100 '
110 '   Print form and set up for input
120 '
130 PRINT @ 256,VC;' Clear user screen and position cursor
140 PRINT "Enter first name" TAB(18) ":"
150 PRINT "Enter last name" TAB(18) ":"
160 PRINT "Enter address" TAB(18) ":"
170 PRINT "Enter city" TAB(18) ":"
180 PRINT "Enter state" TAB(18) ":"
190 PRINT "Enter zip" TAB(18) ":"
200 '
210 '   Use Input@ to get information
220 '
230 PT%=256 : LI%=20 : GOSUB 1000' Get first name
240 IF NOT(SI=VC OR SI=VR) THEN S1=SI' Valid input?
250 PRINT @ PT%+20, VL ; S1 ;' Display it
260 IF SI=VC THEN PT%=512 : GOTO 480' Wrap around on CLEAR
270 '
280 PT%=PT%+64 : LI%=30 : GOSUB 1000' Get last name
290 IF NOT(SI=VC OR SI=VR) THEN S2=SI' Valid input?
300 PRINT @ PT%+20, VL ; S2 ;' Display it
310 IF SI=VC THEN 230' If CLEAR go back
320 '
330 PT%=PT%+64 : LI%=30 : GOSUB 1000' Get address
340 IF NOT(SI=VC OR SI=VR) THEN S3=SI' Valid input?
350 PRINT @ PT%+20, VL ; S3 ;' Display it
360 IF SI=VC THEN PT%=PT%-128 : GOTO 280' If CLEAR go back
370 '
380 PT%=PT%+64 : LI%=30 : GOSUB 1000' Get city
390 IF NOT(SI=VC OR SI=VR) THEN S4=SI' Valid input?
400 PRINT @ PT%+20, VL ; S4 ;' Display it
410 IF SI=VC THEN PT%=PT%-128 : GOTO 330' If CLEAR go back
420 '
430 PT%=PT%+64 : LI%=2 : GOSUB 1000' Get state
440 IF NOT(SI=VC OR SI=VR) THEN S5=SI' Valid input?
450 PRINT @ PT%+20, VL ; S5 ;' Display it
460 IF SI=VC THEN PT%=PT%-128 : GOTO 380' If CLEAR go back
470 '
```

```
480 PT%=PT%+64 : LI%=12 : GOSUB 1000' Get zip
490 IF NOT(SI=VC OR SI=VR) THEN S6=SI' Valid input?
500 PRINT @ PT%+20, VL ; S6 ;' Display it
510 IF SI=VC THEN PT%=PT%-128 : GOTO 430' If CLEAR go back
520 IF SI=VR THEN 550' End on ENTER
530 GOTO 230' Continue editing
540 '
550 PRINT : PRINT : PRINT "That's how it's done!" : END
997 '
998 '   Subroutine to input data using Input@
999 '
1000 INPUT @ PT%+20,LI%,"$";SI : RETURN
```

Notice first of all that there are a couple of my own small programming idiosyncrasies in the text. In line 70, I define certain control characters so that I don't have to refer directly to them later. I use "S" and "V" as string variables. "S" because it is the first letter of <s>tring and "V" for <v>ideo control codes. Also, "I" through "N" are integers. This is left over from the good old FORTRAN days.

Now to the technique. I think it can be outlined like something like this :

I. Clear screen and print input form

II. Input fields
    - A. Set input position
    - B. Set input length
    - C. Input data

III. Interrogate input
    - A. Is it ENTER or CLEAR?
    - B. If not, store data otherwise act

IV. Display data

V. Move forward or back

Let's take an example from the program. Look at lines 230-260. In lines 130-190, we've alreay done step one. That is, we cleared the screen (as much of it as we will use) and printed the form for input.

In line 230, we execute step two. We set "PT%", which is our print position, to 256. We set "LI%", which is our length of input, to 20. Then we execute a gosub to line 1000 which actually calls Input@.

Once the input is returned; in line 240 we examine it. The statement there checks to see if the input variable is equal to ENTER or CLEAR. If it is not, then the permanent variable is set equal to the input variable as valid data is assumed. If the data is ENTER or CLEAR, then permanent variable remains unaffected.

In line 250, we display the current permanent variable. This will either be what was just entered, or if what was just entered was a control key, then it will be whatever was set for that variable before.

In line 260, notice that we are checking to see if it is CLEAR only. If they pressed CLEAR, then we back up a prompt. If it wasn't CLEAR, then we assume either ENTER or else valid data, and in either case we advance a prompt.

So the principle is: use an intermediate variable to get input. Check this variable for control keys before altering your permanent variable. Display only the permanent variable (which should always contain valid data). Use ENTER and CLEAR to page up and down the screen.

Enter this sample program and try it for yourself. It works well and looks smooth! I encourage you to use similar techniques in YOUR programming. Let's clean up that screen!

-Ed.

Random Routines

by Todd N. Tolhurst

This month, we are pleased to present two programs, one in BASIC, and the other in assembly language. The BASIC program, COMPARE/BAS, was written by Hile Troy of Washington, D.C. Hile works for the Department of Defense, and is interested in TRS-80 programming and its applications in forestry.

COMPARE/BAS

COMPARE/BAS is a BASIC program that will compare two files byte-for-byte, and print any mismatches on the screen. COMPARE will run under DOSPLUS 3.3, 3.4, or 4.0. When using the program, bear in mind that it requires 2 file buffers, so enter BASIC using the command "BASIC -F:2" and then run COMPARE/BAS. The author, Hile Troy, recommends that you compile the program for maximum speed. The program will compile without change under Microsoft's BASCOM, and should run with little modification under BASIC/S or ACCEL.

```
10 '
20 '    File COMPARE program
30 '
40 '    For DOSPLUS 3.3/3.4/4.0
50 '
60 '    Written 23-May-82
70 '    Author: Hile Troy
80 '            Washington, D.C.
90 '
100 '
200 CLEAR 1000:DEFINT I,J,K:DEFSTR S,T
205 HX$="0123456789ABCDEF"
210 CLS:PRINT"COMPARE - File compare utility 1.4"
220 PRINT"By Hile Troy":PRINT
290 '
300 LINEINPUT"Filespec 1: ";S1 :'get first filename
310 LINEINPUT"Filespec 2: ";S2 :'get second filename
315 '
320 S=S1:GOSUB 10000:'look for file
330 IF JF THEN PRINTS1;" not found!" : PRINT : GOTO 300 :'file 1
not found
340 '
350 S=S2:GOSUB 10000:'look for file
360 IF JF THEN PRINTS2;" not found!" : PRINT : GOTO 300 :'file 2
not found
390 '
400 OPEN"R",1,S1 :'open files for compare
410 OPEN"R",2,S2
420 FIELD 1,1 AS SY:FIELD 2,1 AS SZ
430 J1=VARPTR(SY):J1=PEEK(J1+1)+PEEK(J1+2)*256-32:'file 1 dcb
440 J2=VARPTR(SZ):J2=PEEK(J2+1)+PEEK(J2+2)*256-32:'file 2 dcb
```

```
450 LA=(PEEK(J1+12)+PEEK(J1+13)*256)*256+PEEK(J1+8):'file 1
length
460 LB=(PEEK(J2+12)+PEEK(J2+13)*256)*256+PEEK(J2+8):'file 2
length
470 IF LA=>LB THEN LC=LA ELSE LC=LB:'select longest file length
490 '
500 I1=0:'beginning record #
505 L0=0 : 'current byte#
520 I1=I1+1:GET1 : GET2:'get a record from each file
525 I2=0:'beginning byte# within rec
530 FIELD1, (I2) AS SY, 1 AS SA:'field for current byte
540 FIELD2, (I2) AS SZ, 1 AS SB
550 IF SA=SB THEN GOTO 700
590 '
600 PRINT"Record ";
610 I=I1/256::GOSUB 12000:PRINTSH;:I=I1-I*256:GOSUB
12000:PRINTSH;
620 PRINT", Byte ";
630 I=I2:GOSUB 12000:PRINTSH;
640 PRINT" - 1:";
650 I=ASC(SA):GOSUB 12000:PRINTSH;
660 PRINT", 2:";
670 I=ASC(SB):GOSUB 12000:PRINTSH
690 '
700 I2=I2+1
705 IF (I1-1)*256+I2=LC THEN END:'if end-of-file
710 IF I2<256 THEN GOTO 530 ELSE GOTO 520
9990 '
10000 ON ERROR GOTO 10500:'if error
10005 JF=0:'reset flag
10010 OPEN"I",1,S:'attempt to open file
10020 CLOSE 1:ON ERROR GOTO 0:RETURN
10500 JF=-1:RESUME NEXT
12000 IH=I/16:IL=I-IH*16:'get msn and lsn
12010 SH="":'init hex$
12020 I=IH:GOSUB 12100:'convert msn
12030 I=IL:'convert lsn
12100 SH=SH+MID$(HX$,I+1,1)
12110 RETURN
```

MKEY/CMD

Did you ever wish that you could reduce oft-typed lines such as "BASIC -F:3", "FOR I=0 TO 2000:NEXT", "CONVERT /CMD:1 :0 (V13)", or "10 REM Copyright 1982, by John Doe, Esq." to just a couple of keystrokes? Well, this month's program, MACRO-KEY, does just that.

MACRO-KEY is a machine-language program that inserts itself into the TRS-80's keyboard driver, and allows up to 255 pre-defined characters to be assigned to each key on the TRS-80's keyboard. The program works equally well on Model I or Model III (if you are using a Model I, change the value of HIGH3 from 4411H to 4049H). Type the program listing (Fig. 1a & 1b) into a Z-80 assembler (we used M-ZAL, you can use anything that's handy), and assemble it under the name MKEY/CMD.

MKEY/CMD gets its "key definitions" from a data file that you specify when you execute MKEY. To generate this "key file", you must use the KEYGEN program. Assemble the listing (Fig. 2) and save it under the name KEYGEN/CMD.

To create your key definitions, type KEYGEN, followed by the name of the definition file you wish to create, from the DOSPLUS command mode, like this:

```
KEYGEN TESTKEYS:1
```

When the ">" prompt appears, you may define your keys something like this:

```
>B BASIC -F:1;
>C CAT :1 (T);
>D DIR :
>L LLIST;
>F FORMS (P=66,L=60,W=80,NULL=N);
>A CLEAR (RESET);
```

The general syntax goes like this:

<key> <space> <macro-string>

If you wish a carriage return to appear in the line, use a semicolon, ";", in its place. KEYGEN will "translate" the semicolon into a carriage return for you. When you have entered all of the keys that you wish to define, press <BREAK> to exit KEYGEN.

To activate MKEY, all you need do is type the program name followed by the name of the key definition file, like this:

```
MKEY BKEYS
```

This will load the key definitions from the file "BKEYS" and install the MKEY driver. To use MKEY, you press the <CLEAR> key followed by any key that you have defined in the key definition file. If you wish to use the <CLEAR> key (to clear the screen, for instance), you must press <CLEAR> twice.

MKEY may be disabled by using the DOSPLUS command CLEAR (RESET).

```
00010 ;
00020 ;       MACRO-KEY
00030 ;
00040 ;       CREATED : 06/24/82  TNT/MSS
00050 ;       UPDATED :   /  /
00060 ;
00070 KIDCB   EQU     4015H           ;*KI DCB
00080 HIGH3   EQU     4411H           ;TOPMEM POINTER
00090 OPEN    EQU     4424H           ;OPEN VECTOR
00100 ERROR   EQU     4409H           ;ERROR VECTOR
00110 FSPEC   EQU     441CH           ;FSPEC VECTOR
00120 GET     EQU     0013H           ;CHAR READ VECTOR
00130 EOFERR  EQU     28              ;EOF ERROR CODE
00140 BADNAM  EQU     19              ;BAD NAME ERROR
00150 CTRL    EQU     31              ;CTL KEY=CLEAR
00160 ;
00170         ORG     5200H
00180 ;
00190 ;       INSTALL MACRO-KEY DRIVER
00200 ;
00210 START   LD      DE,DCB          ;FILE DCB
00220         CALL    FSPEC           ;PUT NAME IN DCB
00230         JR      Z,START3        ;IF NO ERROR
00240         LD      A,BADNAM        ;BAD NAME ERROR
00250         JP      ERROR           ;POST ERROR
00260 ;
00270 START3  LD      HL,BUFF         ;DISK I/O BUFFER
00280         LD      B,0             ;LRL=256
00290         CALL    OPEN            ;OPEN FILE
00300         JP      NZ,ERROR        ;IF ERROR
00310 ;
00320         LD      HL,MKEYS        ;POINT TO MKEY TABLE
00330 START0  CALL    GET             ;GET BYTE FROM FILE
00340         JR      Z,START1        ;IF NO ERROR
00350         CP      EOFERR          ;END-OF-FILE?
00360         JP      NZ,ERROR        ;DISK ERROR
00370         JR      START2          ;END-OF-FILE
00380 ;
00390 START1  LD      (HL),A          ;FILL TABLE
00400         INC     HL
00410         JR      START0
00420 ;
00430 START2  LD      DE,MKEY         ;PROGRAM START
00440         SBC     HL,DE           ;GET PROG LEN
00450         EX      DE,HL           ;PLACE IN DE
00460         PUSH    DE              ;SAVE PROG LEN
00470         LD      HL,(HIGH3)      ;GET TOPMEM
00480         SBC     HL,DE           ;FIND NEW TOPMEM
00490         DEC     HL
00500         LD      (HIGH3),HL      ;ADJUST TOPMEM
00510         INC     HL
00520 ;
00530         EX      DE,HL           ;PROG START IN DE
00540 ;
00550 ;       RESOLVE REFERENCES TO "CHRPNT"
00560 ;
00570         LD      HL,CHRPNT-MKEY  ;OFFSET FROM START
00580         ADD     HL,DE           ;FIND ADDRESS
00590         LD      (REF0+2),HL
00600         LD      (REF1+2),HL
00610         LD      (REF2+2),HL
00620 ;
00630 ;       RESOLVE REFERENCES TO "MKEYS"
00640 ;
00650         LD      HL,MKEYS-MKEY   ;OFFSET FROM     RT
00660         ADD     HL,DE           ;FIND ADDRESS
00670         LD      (REF3+1),HL
00680 ;
00690 ;       "FILL IN" CALLS TO KBD DRIVER
00700 ;
00710         LD      HL,(KIDCB+1)    ;GET DRIVER ADDR
00720         LD      (MKEY0+1),HL
00730         LD      (MKEY1+1),HL
00740 ;
00750 ;       INSTALL MACRO-KEY DRIVER IN *KI DCB
00760 ;
00770         LD      (KIDCB+1),DE
00780 ;
00790         LD      HL,MKEY
00800         POP     BC              ;GET PROG LEN
00810         LDIR                    ;MOVE INTO HIMEM
00820 ;
00830         XOR     A
00840         LD      (KIDCB+3),A
00850 ;
00860         RET                     ;DONE
00870 ;
00880 DCB     DEFS    32              ;FILE DCB
00890 BUFF    DEFS    256             ;FILE I/O BUFFER
00900 ;
00910 ;       MACRO-KEY DRIVER ROUTINE
00920 ;
00930 MKEY    BIT     0,(IX+3)        ;M-KEY ACTIVATED?
00940         JR      Z,MKEY0         ;NO
00950 ;
00960 REF0    LD      DE,(CHRPNT)     ;POINT TO NEXT CHR
00970         LD      A,(DE)          ;GET CURRENT CHR
00980         INC     DE              ;ADVANCE POINTER
00990 REF1    LD      (CHRPNT),DE     ;STORE POINTER
01000 ;
01010         OR      A               ;GET STATUS
01020         RET     NZ              ;IF VALID CHR
01030         RES     0,(IX+3)        ;DISABLE M-KEY
01040         RET                     ;DONE
01050 ;
01060 MKEY0   CALL    $-$             ;CALL KBD DRIVER
01070         CP      CTRL            ;CONTROL KEY?
01080         RET     NZ              ;NO
01090 ;
01100 MKEY1   CALL    $-$             ;GET MACRO-KEY
01110         JR      Z,MKEY1         ;WAIT FOR KEY
01120         CP      CTRL            ;CONTROL KEY?
01130         RET     Z
01140 ;
01150         CP      'a'             ;LOWER CASE?
01160         JR      C,MKEY7         ;NO
01170         CP      'z'+1           ;LOWER CASE?
01180         JR      NC,MKEY7        ;NO
01190         RES     5,A             ;IGNORE CASE
01200 ;
```

Fig. 1a (cont)

```
01210 MKEY7   EX      DE,HL           ;PRESERVE HL REG
01220 REF3    LD      HL,MKEYS        ;START OF TABLE
01230 MKEY8   INC     (HL)            ;END-OF-TABLE?
01240         DEC     (HL)
01250         JR      Z,MKEY4         ;NOT IN TABLE
01260 ;
01270         CP      (HL)            ;IS THIS IT?
01280         JR      Z,MKEY3         ;YES
01290 MKEY9   INC     (HL)            ;ADVANCE TO NEXT
01300         DEC     (HL)            ;=0?
01310         INC     HL
01320         JR      Z,MKEY8         ;TRY NEXT ENTRY
01330         JR      MKEY9
01340 ;
01350 MKEY3   INC     HL              ;=>MACRO STRING
01360         EX      DE,HL           ;SWITCH REGS BACK
01370 REF2    LD      (CHRPNT),DE     ;POINT TO STRING
01380         SET     0,(IX+3)        ;ACTIVATE M-KEY
01390 ;
01400 MKEY4   XOR     A               ;SET ZERO STATUS
01410         RET
01420 ;
01430 CHRPNT  DEFW    0
01440 MKEYS   DEFB    0
01450 ;
01460         END     START
```

Fig. 1b

```
00010 ;
00020 ;       MACRO-KEY FILE GENERATOR
00030 ;
00040         ORG     5200H
00050 ;
00060 KEYIN   EQU     40H
00070 DSP     EQU     33H
00080 DSPLY   EQU     4467H
00090 FSPEC   EQU     441CH
00100 INIT    EQU     4420H
00110 CLOSE   EQU     4428H
00120 PUT     EQU     1BH
00130 ERROR   EQU     4409H
00140 CR      EQU     0DH
00150 ;
00160 START   PUSH    HL              ;SAVE COMMAND LINE
00170         LD      HL,TITLE        ;PRINT PROGRAM
00180         CALL    DSPLY           ;TITLE ON SCREEN
00190         POP     HL              ;RESTORE POINTER
00200 ;
00210         LD      DE,DCB          ;FILE DCB
00220         CALL    FSPEC           ;MOVE NAME TO DCB
00230         LD      HL,BUFF         ;FILE I/O BUFFER
00240         LD      B,0             ;LRL=256
00250         CALL    INIT            ;CREATE FILE
00260         JP      NZ,ERROR        ;IF ERROR
00270 ;
00280 GETLIN  LD      A,'>'           ;PRINT LINE PROMPT
00290         CALL    DSP
00300         LD      HL,KBUFF        ;KEYBOARD BUFFER
00310         LD      B,-1            ;INPUT LENGTH
00320         CALL    KEYIN           ;GET INPUT
00330 ;
00340         JR      C,DONE          ;IF DONE
00350         INC     B               ;ANY CHARACTERS?
00360         DEC     B
00370         JR      Z,GETLIN        ;TRY AGAIN NULL
00380 ;
00390         LD      B,-1            ;FLAG 1ST CHR
00400 ;
00410 WRTLIN  LD      A,(HL)          ;GET A CHR
00420         INC     B               ;TEST FLAG
00430         JR      NZ,WRT4         ;IF NOT 1ST CHR
00440         INC     HL              ;SKIP NEXT CHR
00450         JR      WRT3            ;WRITE TO FILE
00460 ;
00470 WRT4    CP      CR              ;CARRIAGE RETURN?
00480         JR      NZ,WRT2         ;NO
00490         LD      A,0             ;TRANSLATE TO NUL
00500 ;
00510 WRT2    CP      ';'             ;SEMICOLON?
00520         JR      NZ,WRT3         ;NO
00530         LD      A,CR            ;TRANSLATE TO C/R
00540 ;
00550 WRT3    LD      DE,DCB
00560         CALL    PUT             ;WRITE BYTE TO FIL
00570         JP      NZ,ERROR        ;IF ERROR
00580         OR      A               ;END-OF-LINE?
00590         JR      Z,GETLIN        ;IF DONE WITH LINE
00600         INC     HL              ;NEXT BYTE
00610         JR      WRTLIN          ;WRITE MORE CHR'S
00620 ;
00630 DONE    LD      DE,DCB
00640         LD      A,0             ;END-OF-TABLE
00650         CALL    PUT             ;BYTE INTO FILE
00660         CALL    CLOSE           ;CLOSE FILE
00670         RET                     ;RETURN TO DOS
00680 ;
00690 TITLE   DEFB    28
00700         DEFB    31
00710         DEFM    'MACRO-KEY - File generation '
00720         DEFM    'utility 1.0'
00730         DEFB    10
00740         DEFM    'DOSPLUS News Information Center '
00750         DEFM    'Jul/Aug 1982'
00760         DEFB    10
00770         DEFB    13
00780 ;
00790 DCB     DEFS    32
00800 BUFF    DEFS    256
00810 KBUFF   DEFS    256
00820 ;
00830         END     START
```

Fig. 2

Florida Micro Computer Systems

HARDWARE
SYSTEMS ANALYSIS
Custom Applications Software

1631 West McNab Road
Ft. Lauderdale, Flc
Phone 971-9300 33309

BASESCRIPT + YOU +  = SPEED
RELIABILITY
EASE OF USE
FLEXIBILITY

THE BASESCRIPT SYSTEM

The Basescript system is more than just another software program, it's a whole new concept in program packaging. Now you can buy a work processing package at a FAIR price that includes everything you need in the typical office or home environment. The system's modules, and their capacities and functions are outlined below:

1. The **BASESCRIPT** work processor: This full screen editor type of work processor is both easy to use and full featured enough to meet the needs of just about anyone using a word processor. Some of it's features are:

** Insert character(s)
** Insert Lines
** Delete Character(s)
** Delete Lines
** Delete Block
** Columnar MATH function
** Underlining
** **Boldfacing**
** True Tabbing Function
** Left/Right justification
** Page numbering
** Single sheet or tractor feed
** All print features available to any printer that supports a backspace code (08 decimal)
** Block insertion markers (superior form of global search and replace)
** HELP command with full on screen explanations of all special command functions.
** Automatic centering and margins on any size paper up to and including legal length.
** Written in BASIC & Machine code for ease of modification and speed.
** Italics and emphasized print for EPSON/GRAFTRAX.

2. The **BASESCRIPT** Forms production program: This program is used in conjunction with user created form documents. It allows fast fill out and update of forms created on the BASESCRIPT work processor by automatically detecting and jumping to pre-defined insertion points on the form.

3. The **BASESCRIPT** Mailing List programs: This set of programs create and maintain an alphabetic and zip code sorted mailing list of up to 2500 names. The programs within the module support the following functions:

Florida Micro Computer Systems

HARDWARE
SYSTEMS ANALYSIS
Custom Applications Software

1631 West McNab Road
Ft. Lauderdale, Florida
Phone 971-9300 33309

** Automatic creation of "Clean" data disks on 35, 40, and 80 track disk drives (single or double headed) and/or hard disk drive systems.
** Add to the mailing list.
** Delete from the mailing list.
** Define and assign files to (up to) eight user defined sub-lists.
** Print lists of entries (Alphabetic, Zip, or Numeric order).
** Print labels &/or envelopes (tractor or single feed) with user defined selectivity by zip code &/or sublist category.
** Full screen editing (correcting) of files.
** Inquiry by last name or file number.
** Zip code report generator (# of files within given zips).

4. The **BASESCRIPT** Custom Document Printing Program: This program allows the user to MERGE data from the mailing list files into form letters and other similar documents to create "personalized" documents. Printing is selectable by any combination of zip codes and/or user defined sublists. Insertion fields can be user or machine input and include such fields as:

** Last (or Company) name.
** First Name.
** Title and first name.
** Preferred title addressing.
** Street address.
** Town, state, and zip code in any combination or singly.
** Corporate form of address (Sirs, Gentlemen, Ladies, etc.).
** Phone number or &/or other comment element.
** Any string of data in user input mode (replaces global search and replace function).

There are many other functions and features within the system that allow complete and flexible entry, storage, retrieval, and updating of all types of user created documents. The best part of the system, however, just may be its price. The complete package, including a 43 page manual in a hard cover 3 ring binder (with a technical section for those users desiring to modify the programming) sells for just $99.95 to existing owners of the DOS PLUS operating system. For non DOS PLUS owners, the system sells for just $229.95 INCLUDING a complete DOS PLUS version 3.4 (regular $149.95 value) and DOS PLUS manual. The BASESCRIPT system will operate on TRS-80 MODEL I, MODEL III, and LNW80 microcomputers with 48K of memory and two disk drives (or hard drive).

MOVE UP TO THE BEST, for **LESS**, the **BASESCRIPT** system is all you will ever have to use to satisfy your work processing and mailing list needs.

Patches, fixes, etc...

This issue we have only one patch, but it IS a good one. As I'm sure you aware by now, Scripsit 3.2 functions just fine under DOSPLUS without any patches. That is, as long as you don't :

(1) Transfer old text files from TRSDOS.

or

(2) Want to use text files with anything else.

You see, Scripsit under DOSPLUS was saving the text files with an incorrect end-of-file. It was saving one extra sector at the end of the file. This sector of "trash" caused no problems within Scripsit, because Scripsit uses a "00" byte for its own internal end-of-file marker.

However, when you would convert old text files over from TRSDOS, every so often one of them came with the end-of-file set in a certain manner that Scripsit under DOSPLUS couldn't load it. Scripsit under DOSPLUS would decrement the sector count by one, but all would still be fine, because we had the extra sector of "trash". But when the end-of-file was exact, decrementing the sector count caused it to miss data at the end of the file. This patch will correct that.

Also, for some applications, you may want to create text files saved in standard ASCII and use them later from another program. Because of the extra sector of trash, this was not possible. You ended up with more that you asked for when you went to read it back in. This patch will correct that, also.

Essentially, this patch will cause Scripsit 3.2 when running under DOSPLUS, to save with an exact end-of-file. This patch is for Model III only at this point. We are examining the Model I version to determine whether a similar patch will be needed or not. At this time, though, Model III users appear to be the only ones with a problem.

Apply this patch to the file SCRIPSIT/CMD using DISKDUMP

Type : DISKDUMP SCRIPSIT/CMD and press ENTER

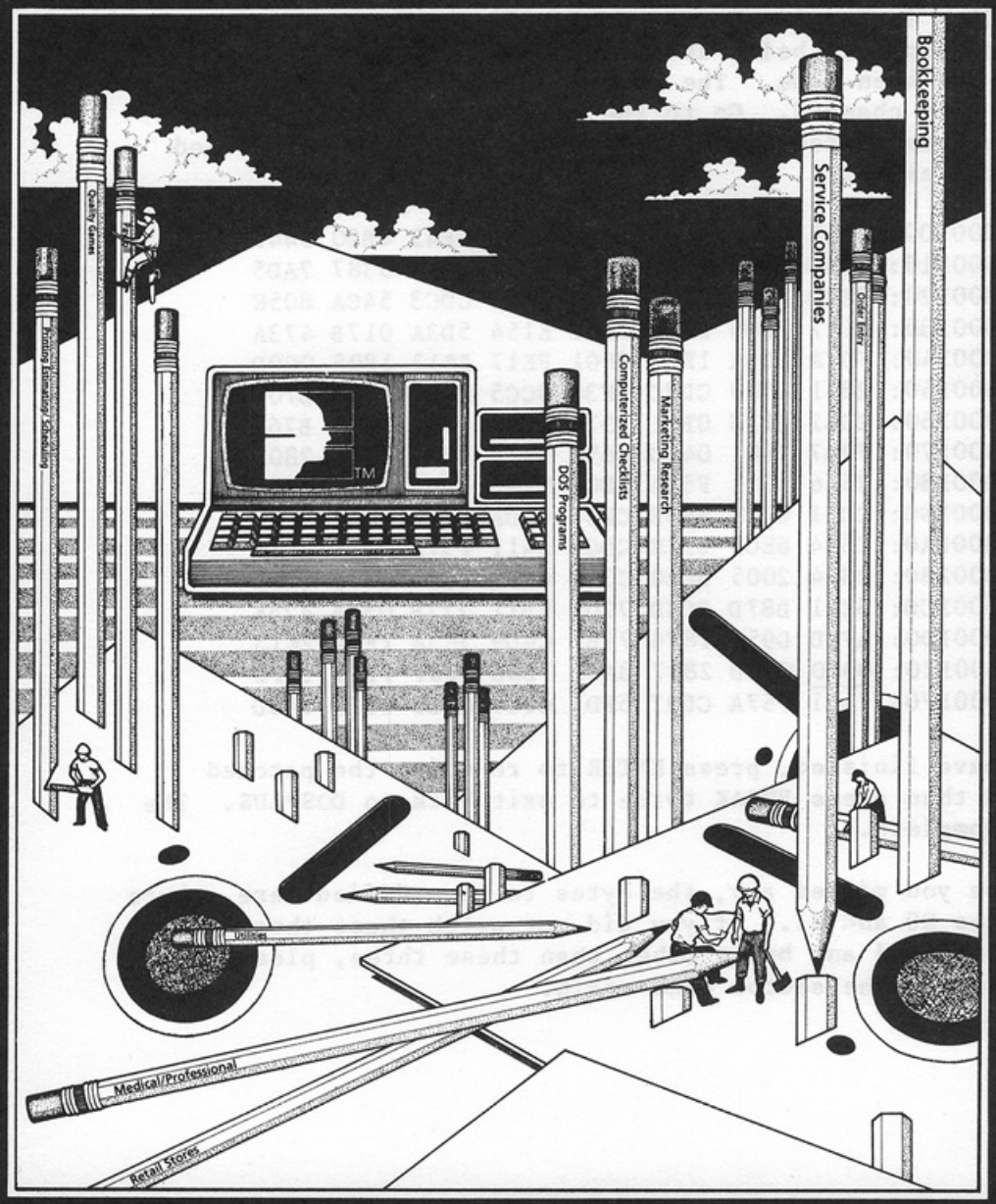
When the first sector is displayed on the screen, type GB and press ENTER. Sector 0B should now be displayed on the screen. The sector displayed below is sector 0B of SCRIPSIT/CMD after the patch has been made. The underlined bytes are the bytes that have been changed. Go to the modify mode by pressing "M" and change those bytes in your sector to match the underlined bytes in the example.

```
000B00: 4E41 4D45 204F 4620 424C 4F43 4B20 544F
000B10: 2045 5843 4841 4E47 453F 20ED 5387 7AD5
000B20: 7ECD C454 11F5 7AC4 1C44 CDC3 54CA 805E
000B30: 2127 7B06 00CD 0D5F E154 5D3A 017B 473A
000B40: FD7A 4F3A 1F44 FE01 FE17 5E13 1805 0C0D
000B50: 2801 0509 CD7C 6F38 0CC5 ED4B A17A B705
000B60: ED42 C138 0B11 F57A CD07 5F3E 01C3 B76F
000B70: 79B7 2801 04CD C65E 10FB EB09 79B7 2801
000B80: 2536 0011 F57A CD07 5FFD CB34 C62A 877A
000B90: CDC1 63CD 646E CDC4 6EDA D663 FDCB 3486
000BA0: C364 6EC5 D57E CDC4 5411 F57A C41C 44CD
000BB0: C354 2005 3E10 C3B7 6F06 0021 277B CDFD
000BC0: 5E11 B87D CDED 7203 EB11 277B 05FA A25E
000BD0: AFCD D95E 18F6 79B7 C4D9 5E3A 1F44 FE13
000BE0: 0000 0C0D 2807 3A01 7B3D 3201 7B79 32FD
000BF0: 7A11 F57A CD07 5FD1 C1C9 C5D5 11F5 7ACD
```

When you have finished, press ENTER to re-write the patched sector and then press BREAK twice to exit back to DOSPLUS. The patch is complete.

In case you missed any, the bytes to be modified were : byte 18 and bytes E0 and E1. If you did not patch these three bytes or if you patched any bytes other than these three, please go back and review the sector once again.

WHEN IT COMES TO SELLING SOFTWARE . . . WE GET TO THE POINT.



JOIN THE SOFTWARE AUTHORITY

Writing computer programs is what you're good at. But no matter how good your programs are, they won't sell themselves. Someone has to take the time and effort to market your works. To make them sell. Selling . . . that's what we're good at.

At Advanced Operating Systems, we're looking for high-quality software. We'll take your finished programs and relieve you of the hassles involved with publishing and distributing, giving you more time to sharpen your skills as a programmer.

We're proud of our team of authors and if you have what it takes, we'd like to add your name to our team. When you team up with Advanced Operating Systems, you can expect professional yet personal attention from our expert staff.

Royalties have never been treated so well. If your program is accepted, we provide highly-competitive royalties and an unequaled marketing effor In addition, we offer you the secur of dealing with a company that will reliably send your royalty checks.

As a division of Howard W. Sams, a wholly-owned subsidiary of ITT, we carry a reputation for excellence in technical service. Let us put this reputation to work for you today. Our 150 representatives in the field will sell your product both to computer stores and directly to the consumer. Your program will benefit from attractive package design, effective copy, and professional layout design.

TYPES OF PROGRAMS WE'RE LOOKING FOR

Applications for:
Retail Stores
Service Companies
Medical/Professional
Quality Games
Bookkeeping
Order Entry
Payroll
DOS Programs
Computerized Checklists
Marketing Research
Printing Estimating/Scheduling

WE ARE CURRENTLY REVIEWING PROGRAMS COMPATIBLE WITH:

Apple
IBM
TRS-80
CP/M
Atari
Commodore
Sinclair
Osborne

■ Contact us and we'll send you an informative brochure, along with a questionnaire and non-disclosure form. Call or write:

Palmer T. Wolf

ADVANCED OPERATING SYSTEMS
450 St. John Road
Michigan City, IN 46360
(800) 348-8558 In Indiana Call:
(219) 879-4693

New Product Announcements

This following section will introduce to you several new products designed to run with DOSPLUS 3.4 or 4.0. If you have a program that runs with DOSPLUS, and you wish to see it advertised in this section; the procedure is this :

(1) Send me a review copy of the program with documentation (only quality software will be displayed here!). This is not necessary if I already have a copy or am familiar with the program.

(2) Send me a text file containing the advertisement you wish to see printed. I reserve the right to edit any advertisement, but I will never print it in edited form unless you get to see and approve a copy first. Please send a file, as I simply do not have time to re-type a two or more page product description.

Remember, these programs are all certified by the authors or publishers to work with DOSPLUS. We are pleased at the ever greater number of software authors and major publishing houses that are making their programs compatible with DOSPLUS. We are more than happy to provide this space to such businesses to advertise their fine software to all you registered DOSPLUS owners reading this newsletter. You are their best market.

INTRODUCING THE VEHICLE PROGRAM

By Kyle Dunn and Karen Story

Even if the cost of gas isn't going up, there's a program suited for DOSPLUS users that keeps track of vehicle maintenance, driver's mileage, gas consumption, as well as repair costs, operation costs, and even the depreciation of the vehicle.

Dunn Write Software Corporation has created the VEHICLE PROGRAM which will be distributed by Micro-Systems Software.

Adaptable for floppies or hard drive, this program will print current, monthly, and yearly reports of Drivers' Miles, Fuel Consumption, Repair/Maintenance, Total Operation Cost, History Comparison, and Depreciation Schedules.

A marine surveying operation in Miami is already using the program on their TRS-80 Model III on a 5 megabyte hard drive. With surveyors traveling across the state of Florida, this program provides them with up-to-date reports on each vehicle and each driver. They also keep track of tune-ups, oil changes, and general vehicle maintenance on each car.

Any business using company cars, vans, trucks (large or small), and even motorcycles can use this package. The Vehicle Program will yield reports on how reliable the company vehicle can be.

We have gone to great lengths to make the Vehicle Program extremely easy to use. For those of you who wish your office help could make use of the computer, instead of fighting it, the instruction manual will describe every step necessary for the person with no computer experience. The manual begins with instructions on how to turn on the computer, and guides you step by step, making this program easy for almost anyone to use.

When you receive the Vehicle Program you will first be asked to enter your company's name and address, which will be printed on all reports. Next you will define the repair and maintenance categories most common to your business. This allows the vehicle program to be taylored specifically to your company. We will then "walk you" through Backup Diskette, Format Diskette, Disk Directory, and Free Space Map. These are easy-to-use Utility Programs which allow you to maintain a reliable system.

The Vehicle Program makes it easy for you to enter vehicle and driver data. Features such as using the up arrow to step back through the questions you have answered, without destroying what you have already entered, allows you to catch data errors the first time around and easily correct them. If an error is entered, the edit section allows you to select the data you wish to correct and automatically updates your files. Thanks to the DOSPLUS sorts you can get a vehicle list sorted by description, tag number, or vehicle number, and a driver list sorted by driver name, license number, or driver number for easy reference.

Entering transactions is a snap with the Vehicle Program. The Transaction Sheet allows you to easily record your gas purchases, drivers' miles, and repair and maintenance costs. These sheets are then passed on to the computer operator to be entered into the computer. All the computer operator must do is answer the questions in the transaction entry section, which follows the transaction sheet filled out by the driver. Once the data has been entered, you can review the operating efficiency of your company vehicles and drivers. As you see the repair/maintenance cost percentage increase over the year, you may decide it's time to replace a vehicle or use a less used vehicle in its place.

A comparative look at the number of miles per gallon each vehicle is getting may help you to route vehicles that cost the least per gallon to cover the routes driven the most. Seeing the average number of miles per gallon drop for a vehicle may indicate the need for a tune up.

This program computes the current, monthly, and yearly totals so as you build your data base of statistics over the years, the Vehicle Program will give management a solid tool to make sound judgements and improve the quality and profit of business. This program is currently available through the Micro-Systems Software dealer network.

The "Other" DOSPLUS

Micro-Systems Software Marketing

Unbeknownst to most TRS-80 users, there exists a "second DOSPLUS" in wide use by program authors and software publishing houses. This "other" DOSPLUS is known as TDOS (for Tiny DOS), and if you've ever purchased a copy of Newscript, Maxi-Manager, or any of dozens of other programs, you may be using TDOS.

Since so many DOSPLUS users are actively involved in the TRS-80 industry as software authors and publishers, we'd like to take this chance to introduce you to TDOS.

TDOS was conceived as an inexpensive means of program distribution. Programs cannot be distributed on TRSDOS, at least if you don't want the long arm of Radio Shack's legal department groping in your direction. Same thing holds true for any DOS - you can't just send out copies of an operating system with every copy of your program, unless you've purchased one copy of the DOS for each copy you sell. Even if you can purchase in dealer quanities, that can add $70-$100 to the cost of a program.

Most applications program do not require the power of an operating system as sophisticated as DOSPLUS 3.4. Indeed, most programs just require a DOS that can "boot" the TRS-80 and allow BASIC or machine-language programs to run. TDOS is such a system. TDOS offers the basic features needed by applications programs. A brief list of library commands is given below:

AUTO	DO
BREAK	KILL
CAT	PAUSE
CONFIG (step rates only)	RENAME
COPY	VERIFY

If a program needs to control such things as the printer page width, or the RS232's baud rate, the program may use POKEs or OUTs to accomplish these functions without the need for the FORMS, RS232, or similar commands.

TDOS is available in two versions: Standard and Extended. Standard TDOS contains TBASIC, the same TBASIC that is provided on the full DOSPLUS system. Extended TDOS includes the DOSPLUS Extended Z80 disk BASIC and the extremely powerful CMD"O" BASIC array sort. Extended BASIC allows the execution of DOS commands and machine-language programs from within a BASIC program.

TDOS also includes several utility programs, as follows:

BACKUP
CONVERT (Model III only)
COPY1
FORMAT

TDOS is the ideal distribution system for program developed using the full DOSPLUS system. Compatibility problems are gone forever when you develop a program under DOSPLUS and distribute it on TDOS, since the two systems are virtually identical at the applications program level.

TDOS is, above all, very inexpensive to use. When you use TDOS, the operating system costs only $2-$5.00, depending on the particular TDOS you require.

For more information on obtaining a TDOS license, or for technical information on TDOS, call or write:

Micro-Systems Software, Inc.
4301-18 Oak Circle
Boca Raton, FL 33431

(305) 983-3390

Guest column

This month's guest column, on writing in BASIC and dealing with "round-off" errors, is written aptly enough by one of our area's premier custom BASIC programmers. Dunn-Write Software is known as one of the highest quality, careful software houses currently developing programs. Kyle Dunn, president and chief programmer, is eminently qualified to discuss BASIC programming techniques. We hope to welcome many such articles by similarly qualified writers on equally relevant subjects. And so, without further ado, Mr. Dunn.

BASIC CAN BE BASIC

By Kyle Dunn
Dunn Write Software Corporation

All too often, the language of Basic is misjudged to be inaccurate. A letter I received from a fellow programmer described the round off error inherent in adding double and single precision numbers.

He said, "As you can see, it would not take long in any size business to give the accountant high blood pressure with accuracy like that."

This problem can be a serious one. I once had to patch someone's payroll system so the W-2 totals would not exhibit this rounding error problem. The bookkeeper was erasing the totals and typing them by hand. They were pleased, as I was, to know that Basic could be arranged to solve the problem.

This problem can be exemplified by:

```
A#=1.00:A#=A#+.01:PRINTA#
```

The computer will print:
1.09999999776483

For those of us not requiring scientific accuracy, we would type:
A#=1.00:A#=A#+.01:A#=A#+.005:A#=FIX(A#*100):A#=A#/100

The computer will print:
1.01

To review the steps

1. A#=1.00 - Assign double precision variable value of 1.00 (A# now equals 1)

2. A#=A#+.01 - Add .01 to value of A# (A# now equals 1.009999999776483)

3. A#=A#+.005 - Add .005 to value of A# so any values of .005 or greater will be rounded up, values less than .005 will be rounded down (A# now equals 1.014999999664724)

4. A#=FIX(A#*100) - The fix command chops off everything to the right of the decimal. If you multiply 1.014999999664724 times 100 and remove everything to the right of the decimal you will get 101.

5. A#=A#/100 - When we finally divide A# by 100 you get the correct answer: 1.01.

I hope the accountants of the world will now rest a little easier now that another computer myth has been dissolved.

On a personal note, I sincerely want to thank all the folks at the DOSPLUS NEWS INFORMATION CENTER for their support in offering us all information. May we all grow together.

Kyle, thank you. - Ed.

Everything you wanted to know about DOSPLUS but were afraid to ask

This column is a regular feature of the DOSPLUS NEWS INFORMATION CENTER, and is conducted by Micro-Systems Software's Technical Support Division. We'll try to answer some of the most-asked questions about DOSPLUS and other Micro-Systems products.

Q. When I run a BASIC program under DOSPLUS, I sometimes receive a "Bad file number" error. The program works OK on TRSDOS. What's wrong?

A. Nothing serious. Remember that TRSDOS asks you the old "How many files" question when you enter BASIC, and if you just press <ENTER>, it assumes 3 file buffers. Under DOSPLUS, you must specify the number of file buffer when you enter BASIC from DOS. Use the syntax "BASIC -F:x", where "x" is the number of file buffers the program requires.

Q. I've been trying to use the FORCE and JOIN commands to send all of the printer output form my program to a disk file. When the program is done, the file shows 0 records, and there's nothing in the file when I LIST it. How come?

A. A file that you create by FORCEing or JOINing must be CLOSEd just like any other file. There are two ways to close such a file. The first involves "forcing (or joining) the device to itself"; for instance, FORCE *PR *PR. Another way is to execute a CLEAR (RESET). Of course, CLEAR does a lot of other stuff, so the FORCE or JOIN is more commonly used.

Q. When I CONVERT a file from Model III TRSDOS 1.3, the file seems to be missing some of the data at the end. Why?

A. When Radio Shack introduced TRSDOS 1.3, they broke with their own standards as set up in TRSDOS 1.1 and 1.2, and changed the way that file lengths are stored in the TRSDOS file directory. For this reason you must tell CONVERT when you are using a TRSDOS 1.3 diskette by typing "CONVERT :s :d (V13)", where "s" is the source drive and "d" is the destination drive. When CONVERTing from a TRSDOS 1.1 or 1.2 diskette, do not use the "(V13)" parameter.

Q. When using TRANSFER, CONVERT, or COPY, I seem to get a lot of "CRC errors". What's a CRC error, and why are they happening?

A. Well, first, a CRC error is reported whenever the floppy disk controller (FDC) detects that the data in a disk sector has been damaged. CRC errors can happen for a variety of reasons, including dirty or misaligned disk drives, foreign material on the diskette, damaged diskettes, etc.

If you experience a lot of CRC errors when using the TRANSFER or CONVERT utilities, or when you use the COPY command, it's posible that your disk drives are configured incorrectly. If you are using standard Radio Shack drives, this can't happen, but many firms that are selling "TRS-80 compatible" drives sell them improperly configured for the TRS-80. TRS-80 disk drives should be set up to "load" the read/write head (bring the head into contact with the diskette) when the computer activates the disk drive's motor. Some drives are configured to load the head each time the computer "selects" a specific drive. When performing a drive-to-drive TRANSFER, CONVERT, or COPY, this results in a great deal of clatter from the drive heads constant loading and unloading. If you suspect that your drives may be configured for "head load on drive select", have a good technician re-configure them for "head load on motor on".

And furthermore ...
(The "so there" section)

Well, I do hope that you have enjoyed this issue of the DOSPLUS NEWS INFORMATION CENTER. I felt that it was once again a very good, informative, issue. We have some other things for you yet (including the Crystal Ball Department), so don't stop turning pages until you run out of paper.

DOSPLUS II update -

DOSPLUS II, the exciting new Disk Operating System authored jointly by Micro-Systems Software Inc. and PowerSOFT, is coming along pretty much on schedule. The features are looking better every day and we anticipate being able to release the system to the public on September the first.

DOSPLUS II is the first truly device independant operating system for the Model II Microcomputer. It will offer an unprecendented level of flexibility without sacrificing the famous DOSPLUS user friendliness. The retail price is $249.95. Registered Micro-Systems Software customers will enjoy their usual "previous customer" discount, bringing the price down to $199.95 retail. Generous dealer discounts will also be in effect. If you order (pre-paid) in advance of the release of the system, Micro-Systems Software will pay for express shipping as soon as the system is ready. Advance orders WILL be shipped first.

The real news on this subject is that we have tested DOSPLUS II on the Model 16 (in its Model II emulation mode) and it works great! The system will operate on the Model 16 as soon as it is released. A true 68000 operating system is being considered, but DOSPLUS II will offer an alternative to the current TRSDOS-II. The system provides significant speed increases with major improvements in reliability and convenience.

DOSPLUS II is configurable for almost ALL varieties of disk drives and other peripheral hardware. Its advanced device routing and filtering capacities provide methods of supporting some varieties of hardware that would otherwise have to be passed by. Look for it at a dealer near you in the month of September.

MicroTerm upgrades -

The response to MicroTerm, our new smart terminal program, has been tremendous. We have discovered some defects in the early versions and are therefore upgrading the program. Most of these are cosmetic defects and should not concern the user as to operation of the program.

We are currently on version 1.3 (most of you have 1.1 or 1.2). When we feel that we have reached the optimum system in both design and appearance, we will upgrade ALL registered owner's with the final manual pages and latest diskettes free of charge.

Please return your MicroTerm registration cards as SOON as possible. This is our only method of knowing where you are. Any users who do NOT get their cards in by the time of the upgrade will have to return the Master diskette in order to be updated. Either method is free, but if you just get that card in quickly, we will do all the work for you.

Inquiries on the matter should be addressed to :

Micro-Systems Software Inc.
Technical Support Division
4301-18 Oak Circle
Boca Raton, FL 33431

(305) 983-3390

Using MicroTerm -

There has been some question in regards to certain features pertaining to the operation of MicroTerm and its upload/download capacities. They were :

(1) How do I upload a file with MicroTerm?

(2) Do I need to open my buffer manually before a download ?

The answer to the first question is two-fold. The first answer applies to BASIC programs, the second to anything else.

If you have a BASIC program that you wish to upload, first save that program in ASCII. I suggest that you use an extension of "/ASC" to indicate that this is a special ASCII file used for uploading. To load this file from BASIC will take mush longer that normal and for standard applications will slow you down.

Enter MicroTerm and go to the command menu. Select "L" for "<L>oad buffer from disk". When it asks you for the filename, answer with the name of your ASCII format BASIC program. After loading, the command menu will be re-displayed and the buffer counter incremented to indicate how much space has been used.

Enter the download section of your particular service. Most download sections will send you a prompt when they are ready for the next line. For instance, Micro-80 will send a colon (":") when it is ready. Find out what this service is going to send.

Select "T" from the command menu and answer the "Prompt String?" question with whatever character this is going to send you. Press ENTER to all other prompts. Return to the terminal mode and transmission will begin.

For this to work properly, three things must be true :

(1) You must have loaded the buffer with the file.

(2) You must have entered correctly what prompt will be sent you.

(3) You must be sitting at the prompt for the first line (i.e. the host computer si requesting the first line of upload).

If the file is not a BASIC program, the procedure is still the same. Only you don't save your program in ASCII, you convert it to an ASCII text file with "Fileconv/Cmd". This is all covered in the MicroTerm manual. The file created with Fileconv will be used in place of the BASIC program described earlier.

The answer to the second question is : maybe. If the host computer is going to send a Ctl-R before it downloads, then no, you don't have to open it yourself. If the host system simply starts dumping text your way, you will have to open the buffer manualy and be ready.

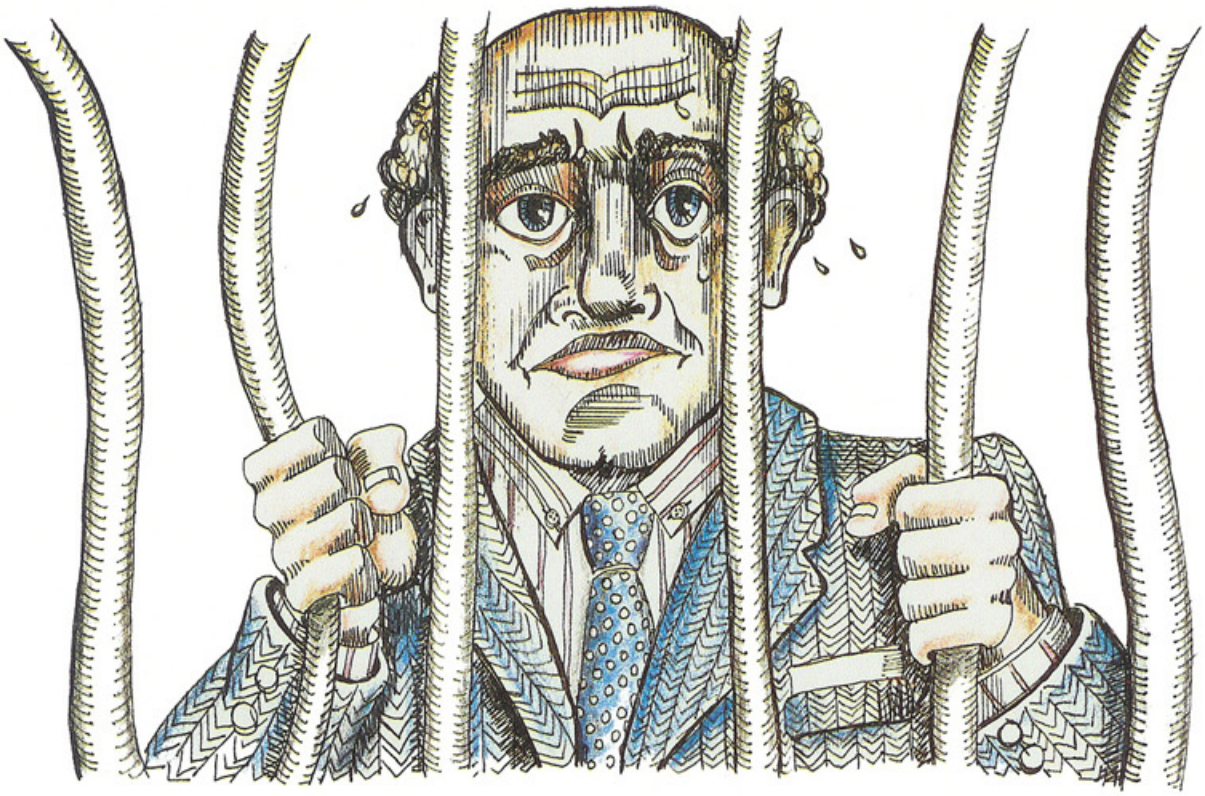
I hope these answers help you some. Remember, as a legitimate purchaser of MicroTerm (or any Micro-Systems product), you are entitled to ask questions and receive answers. We are here to serve you. Thank you.

Crystal Ball Department

Our Word processor has resurrected! We will begin work on it in two to three weeks. We REALLY need a name for it. If you can think of anything, let us know. I will publish the winner's name and suggestion in the newsletter.

I understand that a new version of Pascal 80 was advertised in SoftSide magazine this month. It is DOSPLUS compatible and current owner's may upgrade. Contact Phelps Gates for details.

That's all folks! Remember, this is DOSPLUS country! Onward!
-Ed.



"They shouldn't have touched my Dosplus"

"I only left my keyboard for a few minutes ...when I returned, I found Stamitz from accounting and Miss Sashshay from the secretarial pool fondling my DOSPLUS 3.4. Now if I've told them once, I've told them a hundred times...use my coffee cup. Borrow my key to the employee lounge. Bend my paper clips but, leave my DOSPLUS alone!! Did they listen? Nooooo! Well, I guess I lost my head. Both Stamitz and Sashshay are doing fine. They should be released from the hospital any day now. For me, it's an entirely different story."

Signed,
0076697

Why DOSPLUS creates such fanatically touchy users is not so hard to understand. DOSPLUS 3.4 turns the TRS-80 into something altogether better.

DOSPLUS 3.4 lets your TRS-80 work a lot faster — 5-12 times faster with more accuracy, efficiency and dependability. DOSPLUS 3.4 also has a lot of features that users such as 0076697 find positively endearing. For instance, the ability to read 40 track disks in 80 track drives, and an easy to read operating guide that makes using DOSPLUS as easy as...well, bending a paper clip.

So to increase productivity, to increase the speed and accuracy of your TRS-80. DOSPLUS 3.4 Only $149.95! Still the best DOS for the cost!

P.S. "Get your own DOSPLUS"
0076697

DOSPLUS

DOSPLUS first in quality! First in the industry!

Call Toll Free 1-800-327-8724 ext. 207

MICRO SYSTEMS SOFTWARE, INC.
Specializing in the Tandy Line

4301-18 OAK CIRCLE, BOCA RATON, FL 33431
Outside of Florida phone toll free 1-800-327-8724 ext. 207
For Visa/MasterCard/C.O.D. Orders
Toll free lines will accept orders only!
For applications and technical information, call (305) 983-3390 or drop us a card. Dealers inquiries invited. 384

MICRO-SYSTEMS SOFTWARE, INC.

4301-18 Oak Circle
Boca Raton, FL 33431
Telephone: (305) 983-3390
800-327-8724

DATED MATERIAL
Please do not delay!